A Taxonomy of Automatic Differentiation Tools

David W. Juedes*

Abstract. Many of the current automatic differentiation (AD) tools have similar characteristics. Unfortunately, it is often the case that the similarities between these various AD tools can not be easily ascertained by reading the corresponding documentation. To clarify this situation, a taxonomy of AD tools is presented. The taxonomy places AD tools into the *Elemental, Extensional, Integral, Operational,* and *Symbolic* classes. This taxonomy is used to classify twenty-nine AD tools. Each tool is examined individually with respect to the mode of differentiation used and the degree of derivatives computed. A list detailing the availability of the surveyed AD tools is provided in Appendix A.

1 Introduction. Over the past 30 years, the development of automatic differentiation (AD) tools has been driven by the need for the efficient evaluation of exact derivative values and fueled by our ever increasing ability to create such tools. One of the principal motivations for the creation of AD tools has been the need for the exact derivative values of functions used in scientific computations. As scientists have used more sophisticated models in which first- and higher-order derivatives are used, this need has intensified. In atmospheric and oceanographic research, the need for exact derivative values is particularly great.

As the need for AD tools has intensified, advances in the science of programming languages and modern computer algebra systems have made the creation of such tools less formidable. Numerous researchers have taken advantage of these advances by creating powerful AD tools. For example, lexical-analyzer generator tools such as Lex and compiler-compiler tools such as Yacc have made the task of creating new languages or extending old languages much less tedious. Several AD tools have been created with the help of language development tools like Lex and Yacc. The most notable of these AD tools are the FORTRAN precompilers JAKE [Spee80a] and PADRE2 [Kubo90a], and the modeling language AMPL [Four90a].

During the SIAM Workshop on Automatic Differentiation, it became clear that the forces driving AD technology had caused a variety of different tools to be created. Of these various tools, many of them had similar characteristics. Unfortunately, the similarities

*Department of Computer Science, Iowa State University, Ames, IA.

between many of these AD tools were not easily ascertained by reading the corresponding documentation. Clearly, a coherent framework for discussing AD tools was necessary.

In an attempt to create a coherent framework for discussing AD tools, this paper presents a taxonomy of AD tools. The taxonomy presented here is designed to accentuate the convergence of ideas on how to provide automatic differentiation and to emphasize the evolutionary process molding these ideas. Each automatic differentiation tool examined in this paper is classified as either an *Elemental, Extensional, Operational, Integral,* or *Symbolic* AD tool. This classification is based on the level of integration of automatic differentiation provided by the source language.

In §4, the taxonomy is applied to survey twenty-nine AD tools. The framework of the taxonomy is used to discuss the important features of each tool. We primarily examine each tool with respect to the technique (mode) used to evaluate derivatives and with respect to the maximum degree of derivatives that can be computed. Other features are examined when appropriate. These additional features include the storage strategy for the reverse mode and restrictions on the source language. A summary of this examination is provided in Table 1.

Several of the tools have yet to be named by their respective authors. In the remainder of this paper, unnamed tools are referred to by the first author's initials followed by a number. A large percentage of the tools presented are either commercially available, or available free of charge or at a nominal cost. The remaining tools are unavailable. A list detailing the availability of these tools is provided in Appendix A.

2 Preliminaries. *Automatic differentiation* is the process of producing the values of a function's derivatives from some representation of the function. The key difference between classical automatic differentiation and symbolic differentiation is that the later generates the symbolic representation of a function's derivatives while the former does not. Automatic differentiation is primarily concerned with the differentiation of *algorithms* expressed in some general purpose programming language.

We are primarily concerned with *scalar functions* of the form $f : \mathrm{R}^m \to \mathrm{R}$ and *vector functions* of the form $f : \mathrm{R}^m \to \mathrm{R}^n$. Let $X =< x_1, ..., x_m >$ be a vector of length m, $Y =< y_1, ..., y_n >$ be a vector of length n, and $y \in \mathrm{R}$. Then the *gradient* of a scalar function $y = f(X)$ is $G \in \mathrm{R}^m$ where each $G_i = \partial f / \partial x_i$. The *Hessian* of the scalar function $y = f(X)$ is $H \in (\mathrm{R}^m)^m$ where each $H_{ij} = \partial f / \partial x_i \partial x_j$. The *Jacobian* of a vector function $Y = f(X)$ is $J \in (\mathrm{R}^m)^n$ where each $J_{ij} = \partial y_i / \partial x_j$. Note that when we refer to derivatives, we implicitly refer to related values such as Taylor series or sensitivities.

There are essentially two different techniques used to automatically differentiate functions, the *forward* mode and the *reverse* mode. These different techniques can be best described by considering the *directed acyclic graph* (DAG) of the computation $< y_1, ..., y_n >= F(< x_1, ..., x_m >)$ for some vector function F. In the *reverse* mode, derivative values propagate from the dependent variables $y_1, ..., y_n$ to the independent variables $x_1, ..., x_m$. In the *forward* mode, derivative values propagate from the independent variables to the dependent variables. For more in-depth descriptions of these automatic differentiation techniques, see [Rall81a, Irim87a, Grie89a, etc.].

As noted by Griewank [Grie89a], neither the forward mode nor the reverse mode is optimal in all cases. The forward mode can be used to produce the partial derivatives of all dependent variables with respect to a single independent variable in time proportional to the evaluation of F. Similarly, the reverse mode can be used to produce the partial derivatives of a single dependent variable with respect to all independent variables in time proportional to the evaluation of F. These two previous observations can be used to estimate which mode is more efficient (in terms of numeric operations) when computing Jacobians. Consider the function $< y_1, ..., y_n >= F(< x_1, ..., x_m >)$. For values of $n \gg m$, the forward mode is

probably more efficient than the reverse mode. Similarly, for values of $m \gg n$, the reverse mode is probably more efficient than the forward mode.

Current implementations of the forward and reverse mode differ most in terms of spatial complexity. A typical implementation of the forward mode requires less than m (the number of independent variables) times the storage requirement of the original function. Most implementations of the reverse mode require additional space proportional to the number of numeric operations executed by the original function.

3 A Taxonomy. As mentioned earlier, AD tools provide support for producing the values of a function's derivatives from some representation of the function. Each AD tool assumes that this function is represented in some *source* language, which may be a symbolic language or a programming language. AD tools aid in the transformation from the source language to a form which may easily be used to generate derivative values. This transformation may be either explicit or implicit. For example, the FORTRAN precompiler in the GRESS [Horw88a] package does an explicit transformation from its source language (a minor extension of FORTRAN) to standard FORTRAN. The taxonomy presented in this paper classifies automatic differentiators based on how, where, and when this transformation is performed.

The most basic class is composed of the *elemental* AD tools. These tools operate on the premise that every useful function can be decomposed into a sequence of *elementary operations.* The derivative value of an elementary operation is only dependent upon the values of the operation's arguments and their derivatives. Therefore, computing derivatives for the elementary operations can be performed locally and at a fixed cost. Elemental tools are often implemented as a set of procedures, one for each operation. Each procedure takes as input the values of an operation's arguments and its derivatives, and returns the result and its derivatives. The transformation from the source language is performed manually by decomposing a function into a sequence of procedure calls for the elementary operations. Lawson's WCOMP and UCOMP packages [Laws71a] are examples of elemental AD tools.

Based on the concepts pioneered by the elemental tools, the class of *extensional* tools provide extensions to standard programming languages for automatic differentiation. They usually automate the process of decomposing complicated right-hand-sides of equations into sequences of elementary operations. Typically, automatic differentiators in this class use *precompilers* to perform the transformation from the extensions to the original language. For example, the FORTRAN precompilers JAKEF [Hill85a], GRESS [Horw88a] and PADRE2 [Kubo90a] transform their variants of FORTRAN into standard FORTRAN 77. For this reason, we classify them as extensional AD tools.

The *integral* tools push the ideas embodied by the extensional tools one step further. Integral tools have the ability to automatically differentiate functions integrated within their environment or language. Therefore, no explicit transformation of the source code is necessary. The transformation is performed implicitly by the respective compiler or interpreter. Integral tools typically differ from extensional tools in the sense that they are clearly different languages, and not just minor extensions. They are characterized by source languages that directly provide automatic differentiation. Several of the earliest AD tools belong to the integral class. The best examples of these are SLANG [Adam69a, McCu69a, Tham69a] and PROSE [Tham75a, PROS77a, Tham82a, Krin84a, Pfei87a]. More recent examples of integral AD tools are the modeling language AMPL [Four90a], and the FM/FAD [Mazo91a] package.

The *operational* tools have their origins in the ideas embodied by both the elemental and extensional tools. Operational tools provide automatic differentiation within the constructs provided by a programming language that provides *polymorphic* functions and operations. Automatic differentiation is provided by defining a new data type for which differentiation

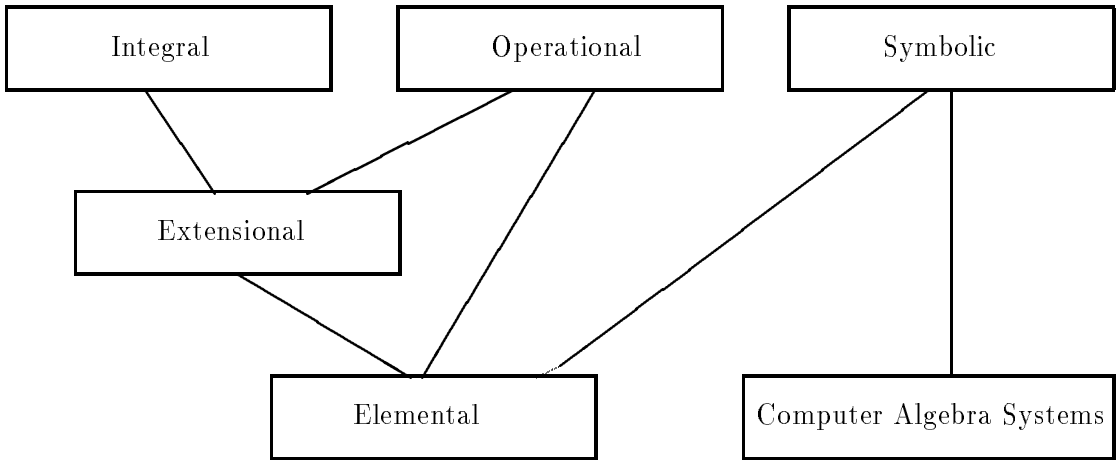

Figure 1: The relationship among the various classes of AD tools.

can be performed easily. For these tools, the transformation necessary to produce derivatives is implicitly handled by the programming language when it interprets the new data type. The standard arithmetic operators are usually *overloaded* to handle operations performed on this new data type. In this case, statements like $x = y*z+sin(z)$ can be written without being decomposed into elementary operations.

The modern programming languages ADA, C++, and PASCAL-SC all provide the operator overloading necessary to create operational AD tools. Some languages, such as C++, also provide *constructors* and *destructors* for variables as they enter and leave their scope of existence. These facilities allow AD tools to transparently allocate and deallocate memory when necessary, as well as perform other maintenance procedures. The operational tools ADOL-C [Grie92a] and MXYZPTLK [Mich90a] make use of these facilities in C++.

The final class of AD tools is the class of *symbolic* tools. Tools in this class either process a symbolic representation of a function at run-time, or use a computer algebra system to produce a differentiated function that is then translated into a standard programming language. These tools are the most difficult to classify and form a diverse class. Most tools in this class have been directly influenced by either computer algebra systems or the elemental AD tools.

The five classes embody the ideas behind a paradigmatic *evolution* of AD tools. The notion that complicated functions can be decomposed into sequences of elementary operations is the basis of all AD tools. This idea is embodied by the Elemental class. Competing ideas on how functions are most naturally represented delineate the remaining classes. The Symbolic class embodies the idea that functions should be represented algebraically instead of algorithmically. The Extensional, Integral, and Operational classes take the opposite stance. These remaining classes are distinguished by their respective ideas concerning the means used to provide automatic differentiation. The Operational class embodies the idea that automatic differentiation can be provided by a differential data type in a language with operator overloading. The Extensional class realizes the notion of extending a language to include automatic differentiation. Finally, the Integral class embodies the idea that automatic differentiation can be provided directly within a programming language. This evolutionary relationship is illustrated in Figure 1.

The five classes provide a sufficient framework for examining a wide variety of AD tools. We subdivide each of the five classes further by the *mode* used to perform automatic differentiation. All current automatic differentiators use either the *forward* or *reverse* mode, and some use both modes. The following section examines twenty-nine AD tools with

respect to their class, mode of differentiation, and degree of derivatives computed.

4 The Survey. The goal of this section is to give a coherent analysis of the current state of AD technology. This goal is accomplished by

(i) using the taxonomy presented in §3 to classify twenty-nine current AD tools, and by

(ii) giving a brief description of each classified tool.

The descriptions stress each tool's mode of differentiation and ability to produce higher derivatives. The descriptions of the tools that fall in the Elemental, Extensional, Operational, Integral, and Symbolic classes are found in §§4.1-4.5, respectively. Additionally, Figures 2 and 3 are provided to emphasize important subdivisions within the Extensional and Operational classes. Table 1 is provided to summarize this section.

The survey presented in this section was generated in the following manner. First, a detailed questionnaire was generated and distributed by the author. The responses to the questionnaire and the available literature were then carefully examined. From this examination, the taxonomy was developed. Following this, each tool was classified, and short, uniform descriptions of each tool were made. The authors of the respective tools were then given the oppurtunity to review the classification and descriptions of their respective tools.

Some of the tools in the survey could have been cross-classified. For example, the GRAD function within MATLAB is classified here as a symbolic AD tool, but it has some of the characteristics common to the integral AD tools. Similarly, FORTRAN CALCULUS is classified here as an extensional AD tool, yet it has some of the properties of the integral AD tools. Tools that could be cross-classified were placed in the class whose characteristics best described them.

4.1 The Elemental Tools. Of the tools surveyed, three of them are elemental tools. All three of these tools produce derivatives in the forward mode.

(1) The **WCOMP** and **UCOMP** [Laws71a, Math89a] packages consist of FORTRAN 77 subroutines for executing operations that propagate derivatives in the forward mode. The WCOMP package contains subroutines that perform operations using real-valued arrays of length $N+1$. Each array contains the value of a function $f(t)$ and and its first N derivatives with respect to an independent variable t. In a similar fashion, the UCOMP package contains subroutines that compute a function's value and its first- and second-order partial derivatives with respect to N independent variables. Each package contains subroutines for a number of elementary binary and unary operations.
A user of either the WCOMP or UCOMP package must directly write code that calls the appropriate subroutines. For example, the statement $a = x + y + z$ is written as follows using the WCOMP package.

```
call SWSUM(N,Y,Z,Q)    // Q=Y+Z
call SWSUM(N,Q,X,A)    // A=X+Q
```

Both packages support single and double precision arithmetic.

(2) The **MJ1** [Jerr89a] package contains the definition of an object for automatic differentiation and its associated operations. The MJ1 package uses Turbo Pascal v5.5 to define an *Ad* object. Each *Ad* object contains a scalar value, gradient vector, and Hessian matrix. The standard unary and binary elementary functions are defined as procedures that operate on *Ad* objects. These procedures propagate derivative values from one object to another in the forward mode.

Complex operations are performed on *Ad* objects by the execution of an appropriate sequence of elementary procedures. For example, the statement $f = x + cos(z * y)$, broken down into the elementary operations $t1 = z*y$, $t1 = cos(t1)$, and $f = x+t1$, is written as follows using MJ1.

```
t1.AdMultiply(z,y) ;
t1.AdCos(t1) ;
f.AdAdd(x,t1) ;
```

(3) The implementation of the **FEED** (the *fast efficient evaluation of derivatives*) algorithm by Tesfatsion *et al.* [Tesf91a] consists of a suite of FORTRAN subroutines that evaluate a function value and its first-, second- and third-order partial derivatives with respect to N independent variables. In the FEED implementation, derivatives are propagated in the forward mode as operations are performed. This implementation is similar to the UCOMP package [Laws71a, Math89a].

4.2 The Extensional Tools. The class of extensional tools contains the largest number (nine) of the surveyed tools. This class consists primarily of *precompilers*. Of the tools in this class, four produce derivatives strictly in the forward mode, one produces derivatives only in the reverse mode, and four use both modes. Figure 2 illustrates this fact.

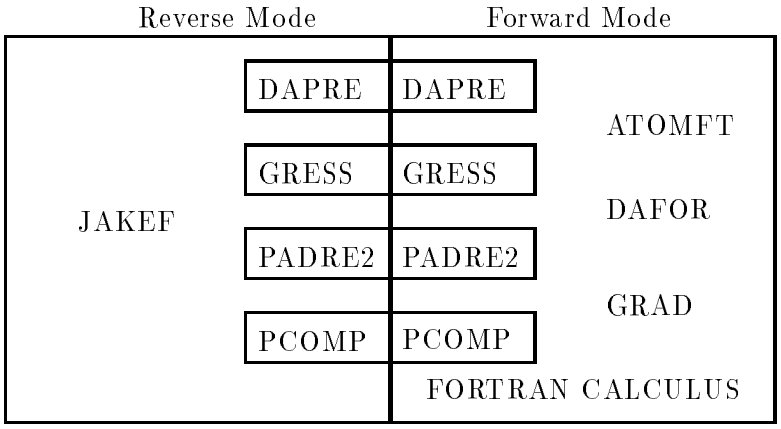

Figure 2: Extensional tools subdivided by differentiation approach.

(4) The general purpose *ordinary differential equation* (ODE) solver **ATOMFT** [Chan82a] is an AD tool in the sense that it provides the automatic generation of code to evaluate Taylor series; however, automatic differentiation is not its primary focus. ATOMFT is a FORTRAN preprocessor that generates FORTRAN source code to solve systems of ODEs. ATOMFT takes as input a system of ODEs written using a FORTRAN-like syntax. The code generated by ATOMFT uses a Taylor series algorithm to solve the system of ODEs. User-defined functions are supported.
ATOMFT reads the system of differential equations and writes FORTRAN source code that uses the recurrence relations derived from each equation to generate the appropriate Taylor series values. The code generated by ATOMFT propagates the Taylor series values strictly in the forward mode. The length of the generated Taylor series is arbitrary and is controlled by user-specified parameters.

(5) The FORTRAN precompiler **DAFOR** [Berz87a, Berz89a, Berz90a] transforms existing FORTRAN code into FORTRAN code that also evaluates derivatives. Partial derivatives can be calculated to an arbitrary-order and with respect to an arbitrary number of independent variables. The transformation creates calls to a library of

subroutines for the elementary operations. These subroutines propagate derivative values in the forward mode.

(6) **DAPRE** [Step91a] is a FORTRAN preprocessor. It takes as input a FORTRAN subroutine and produces a new FORTRAN subroutine in which all of the arithmetic operations are converted to calls to subroutines from a supporting run-time library. The new FORTRAN subroutine can be used in conjunction with other supporting library functions to generate the partial derivatives of the dependent variables of the subroutine with respect to the independent variables of the subroutine. The preprocessor determines which parameters are independent and which parameters are dependent by their respective position in the parameter list of the subroutine. Either the forward mode or the reverse mode can be used to propagate derivative values using DAPRE. (This is done by linking the program with the appropriate run-time library.) Derivatives can be calculated to arbitrary order. The preprocessor puts some restrictions on the subset of FORTRAN used to define the subroutine (see [Step91a]). For example, EQUIVALENCE statements are not allowed. This package was designed to interface smoothly with the NAG library of mathematical software, but it is not NAG specific.

(7) **GRAD** [Garc91a] is a FORTRAN preprocessor. GRAD takes as input a FORTRAN subroutine that computes a function and produces a FORTRAN subroutine that computes the function and its first-order partial derivatives with respect to specified independent variables. Subroutines produced by GRAD can be recycled through GRAD to produce routines for calculating higher-order derivatives. The code that GRAD generates directly computes the partial derivatives and produces Jacobian matrices.

Unlike many other precompilers, GRAD uses no subroutines that represent elementary operations. The partial derivatives are calculated directly and are placed in specially named variables. The code generated by GRAD propagates derivatives in the forward mode.

(8) The **GRESS** (GRadient Enhanced Software System) [Horw88a] package is a system for adding the ability to calculate normalized sensitivities and first-order partial derivatives to existing FORTRAN programs. The GRESS package consists of a FORTRAN precompiler and a library of supporting routines.

The GRESS precompiler produces code that propagates derivatives using either the forward or reverse mode. When the GRESS "Chain" option is selected and a set of independent variables is declared, the GRESS precompiler produces code for calculating the derivatives of specified dependent variables with respect to the declared independent variables. The GRESS "Chain" option produces code that propagates derivatives using the forward mode. When the GRESS "Adgen" option is selected, the GRESS precompiler produces code that calculates the derivatives of the specified dependent variables with respect to every variable initialized by `FORTRAN READ` statements. The GRESS "Adgen" option produces code that propagates derivatives using the reverse mode.

GRESS has been used successfully on a variety of problems. Examples of the application of GRESS can be found in [Horw89a, Horw90a, etc.].

(9) **FORTRAN CALCULUS** [Tham89a, Tham91a] (denoted here by FC) is a commercially available package for mathematical modeling. FC is an extension of FORTRAN 77 that provides macro statements for differentiation, optimization and integration, dynamic arrays, and vector and matrix operations. The FC package consists of a FORTRAN precompiler and a run-time library of supporting routines.

FC's extension to FORTRAN is based on a paradigm referred to as *synthetic calculus* in [Tham82a, Krin84a, Tham89a, Tham91a]. The basic unit within this paradigm is a *model*. Models are subprograms that contain sets of formulas. Differentiation, optimization and integration are performed on specified models. Models may include macro statements for optimization and integration. This allows for the nesting of models.

Differentiation is performed transparently within the optimization and integration processes. The optimization and integration routines use automatically generated partial derivative values from specified models. These partial derivatives are calculated during the execution of the specified model. In order to calculate these derivatives automatically, the FC precompiler converts statements within a model to sequences of subroutine calls. The subroutines propagate the values of variables and their partial derivatives using the forward mode. These subroutines can calculate first- and second-order partial derivatives with respect to an arbitrary number of independent variables.

(10) **JAKEF** [Hill85a] is another FORTRAN precompiler. JAKEF is a version of JAKE [Spee80a] that was developed at Argonne National Laboratory and written using FORTRAN 77. JAKEF takes as input a FORTRAN subroutine defining a scalar or vector function, and generates a FORTRAN subroutine for computing the gradient or Jacobian of the function, respectively. The code generated by JAKEF traces the execution of the function and then uses that trace to propagate derivatives in the reverse mode. The trace is stored in a linearized form in auxiliary integer and real arrays. There is no facility in JAKEF for using auxiliary storage on large traces, although Speelpenning mentions in his unpublished thesis [Spee80a] how such a technique could be implemented.

(11) **PADRE2** [Irim87a, Kubo90a] is also a FORTRAN precompiler. PADRE2 is an extended version of PADRE, which was developed by N. Iwata [Iwat84a]. PADRE2 processes a FORTRAN function or subroutine that evaluates a scalar or vector function and then produces a modified subroutine. This subroutine calculates the original function, its partial derivatives, and an estimate of the rounding error generated. The subroutine executes a sequence of elementary operations for each statement in the original function and builds a computational graph of the execution. The computational graph is later traversed by the subroutine to propagate derivatives and calculate rounding errors. A command line option of PADRE2 determines which mode is used to propagate derivatives.

PADRE2 can be used to calculate first- and second-order partial derivatives. For example, if g is a scalar function with independent variables $X =< x_1, x_2, ..., x_n >$, and $Y =< y_1, y_2, ..., y_n >$ is a vector, then the subroutine generated by PADRE2 can calculate $g(X)$, the gradient $\bigtriangledown g(X)$, the Hessian-vector product $\bigtriangledown^2 g(X) \cdot Y$, and an estimate of the rounding error $\bigtriangleup g(X)$.

The computational graph produced by a PADRE2 generated subroutine is stored exclusively in main memory.

(12) The **PCOMP** [Liep90a] package consists of three independent parts. The first part (a) processes the description of function written in a subset of an extension to FORTRAN, and compiles it into an intermediate code. The second part (b) uses the intermediate code generated by (a) to evaluate the function and its gradient at a single point. Part (b) interpretes the intermediate code and propagates first-order derivatives in the forward mode. The third part (c) uses the intermediate code from (a) to generate FORTRAN subroutines for evaluating the function and its gradient. The subroutines generated by (c) propagate derivatives in the reverse

mode.

4.3 The Operational Tools. Of the tools surveyed, eight are operational tools. Of these tools, six produce derivatives strictly in the forward mode, one produces derivatives strictly in the reverse mode, and one uses both modes. The languages ADA, C++, and PASCAL-SC were used to implement these packages. Two of the packages were implemented using ADA, four using C++, and two using PASCAL-SC. This fact is illustrated in Figure 3.

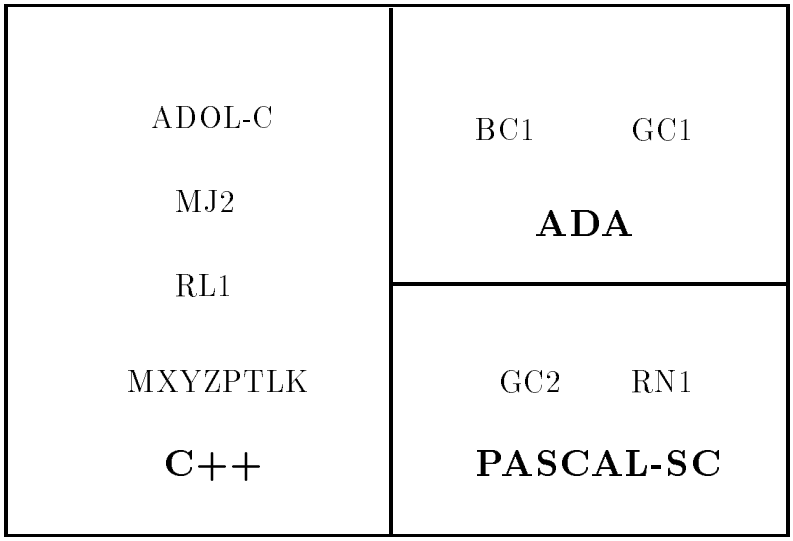

Figure 3: Operational tools subdivided by implementation language.

(13) The **ADOL-C** [Grie92a, Jued90a] package consists of a definition of a class of active variables called *adouble*, and a library of supporting routines. ADOL-C was written using C++, and can be used to differentiate algorithms written in C or C++. With the help of the `f2c` converter, ADOL-C can also handle FORTRAN codes. To differentiate a vector function $< y_1, y_2, ..., y_n >= F(< x_1, x_2, ..., x_m >)$, a user of ADOL-C (i) declares appropriate variables as *adoubles*, (ii) declares the variables $< y_1, ..., y_n >$ to be dependent, and declares $< x_1, ..., x_m >$ to be independent, (iii) traces the computation of F, and (iv) calls routines to generate the partial derivatives. The trace of a computation is used by routines that propagate derivatives in either the forward or reverse mode. First- and higher-order partial derivatives can be calculated by successively propagating derivatives in the forward and reverse mode.
In ADOL-C, arbitrarily nested or recursive functions can be differentiated. Also, the trace of a function is stored sequentially in main memory and is automatically paged to disk when necessary.

(14) The **BC1** [Chri90a] package uses operator overloading in ADA to provide a new data type that aids in the calculation of derivatives. When functions using this new type are executed, the computational graph of that execution is created. This computational graph, which is stored in a linearized form, is used by other routines to propagate derivatives in the reverse mode. The BC1 package can be used to generate first- and second-order partial derivatives (i.e., gradients and Hessians).

(15) **GC1** [Corl91a, Corl91b] is an ADA package for operations on interval valued Taylor series. To generate Taylor series, a user of GC1 writes a program that declares the appropriate variables to be of the *Taylor_type* and uses the overloaded operations provided by GC1. During the execution of such a program, the Taylor series are

calculated in the forward mode automatically. The generated Taylor series could be arbitrarily long. GC1 was designed to be used in an interval ODE solver. It takes special care of applications (like ODEs) in which the Taylor series terms must be completed one term at a time. User-defined functions and procedures involving variables of *Taylor_type* are supported.

(16) The **GC2** [Corl84a] package is written in PASCAL-SC and is used to compute point and interval valued Taylor series operators. GC2 declares the Taylor series data types *Real_Taylor_Type* and *Interval_Taylor_Type*, and overloads all of the PASCAL-SC elementary functions for them. The user of GC2 writes a program using either *Real_Taylor_Type* or *Interval_Taylor_Type* variables. During the execution of the program, the Taylor series of the desired lengths are calculated in the forward mode. User-defined functions and procedures are supported.

(17) The **MJ2** [Jerr89b, Jerr90a] package is very similar to the MJ1 package described in §4.1. MJ2 is written using Turbo C++ v1.0. MJ2 uses operator overloading, and therefore the example code segment $f = x + cos(z * y)$ from §4.1 does not need to be modified.

(18) The **MXYZPTLK** [Mich90a] package is an operational AD tool written in C++. The MXYZPTLK package defines two classes, *DA* and *DAVector*, and provides a library of supporting routines. A number of binary and unary operations are overloaded for these classes (e.g. +, *, sin, cos, etc.). When variables from the two classes are used, derivative values are propagated in the forward mode. MXYZPTLK can be used to calculate arbitrary-order partial derivatives with respect to arbitrarily many independent variables.

(19) **RL1** [Lind91a] is another operational AD tool written in C++. The RL1 package is used to generate derivatives for scalar or vector functions of a single real variable (time). The RL1 package defines a class of tuples which contain the value of a function and its derivatives with respect to a single independent variable. Operations applied to tuples produce tuples of the same form. The standard operations are overloaded for this class. When operations are executed on members of this class, the derivative values are propagated along with the function values in the forward mode. Derivatives up to the third order can be calculated.

(20) The **RN1** [Neid91a] package is an operational AD tool written in PASCAL-SC. RN1 defines a new data type for which the standard operations are overloaded. When operations using this new type are executed, derivatives are propagated in the forward mode. Partial derivatives to arbitrary order and with respect to arbitrarily many independent variables can be calculated. Currently, the author of RN1 considers it to be purely a research project.

4.4 The Integral Tools. Of the tools surveyed, four are integral tools. Of these, one uses the forward mode of automatic differentiation, one uses the reverse mode, and two use both modes.

(21) The **ADDS** (*Automatic Derivative Derivation System*) [Yosh89a] package is an AD tool that defines a new language. The ADDS package provides a language for describing scalar, vector, and matrix expressions, and their partial derivatives. Expressions defined in the ADDS language are translated into FORTRAN. During the translation process, a computational graph of the expression (and alternatively a graph for its partial derivatives) is generated. This computational graph is used to generate the FORTRAN code. The translation process can produce code that calculates partial derivatives in either the forward or reverse mode. The partial derivatives can be calculated to arbitrary order and with respect to arbitrarily many independent variables. ADDS also provides facilities for estimating rounding

errors.

(22) The modeling language **AMPL** [Four90a, Gayd91b] is a declarative language that was designed for expressing mathematical programming problems. Mathematical programming problems are described in AMPL by a sequence of nonlinear expressions. The AMPL translator emits a representation of the DAG for each nonlinear expression given. The DAG can then be used by various *solvers* to evaluate the expression and its gradient at the points leading to a solution. Depending on the solver, the evaluation of gradients is performed using either the forward mode, the reverse mode, or a combination of both modes. The DAG of an expression can also be used to produce FORTRAN or C routines for calculating the expression and its gradient at a point.

(23) The **FM/FAD** [Mazo91a] package is a PC-based set of tools for problem management. Problems (i.e., functions) are defined in FM/FAD using DIFALG, an ALGOL-60 like programming language. Function definitions are *compiled* by the DIFALG compiler and the compiled version is used by the DIFALG interpreter. The *problem solving* aspect of the FM/FAD package uses the DIFALG interpreter directly to calculate function values and their gradients evaluated at a point. The DIFALG interpreter calculates the gradient of the function via the reverse mode. The FM/FAD package only generates first-order partial derivatives.

(24) **FOXY** [Berz90b, Berz90c, Berz90d] is an object-oriented programming language that is similar to PASCAL. FOXY has built in facilities for automatically differentiating functions written in that language. Arbitrary-order partial derivatives with respect to arbitrarily many variables can be calculated. These derivatives are calculated in the forward mode.

4.5 The Symbolic Tools. The class of symbolic AD tools contains five members. Of the members of this class, four use the forward mode of automatic differentiation, while only one uses the reverse mode.

(25) The **AD** [Flan91b] program is a PC-based menu driven program for producing numeric function and derivative values. The AD program allows a function to be entered symbolically. Once entered, the symbolic representation of the function is parsed and a tree representing the function is built internally. The function and its derivatives (up to the 20th order) can then be evaluated at a point. The evaluation process propagates derivatives in the forward mode. Currently the AD package is only a demonstrational tool.

(26) The GRAD [Hill91a] function within the **MATLAB** library of mathematical software can calculate gradients interactively. The parameters to the GRAD function are a character string containing the symbolic description of a function of n variables and a point in n-space (i.e., an array containing n real values). GRAD returns an array of $n+1$ values containing the function value and its gradient evaluated at that point. The character string is parsed internally by the GRAD function, which then uses the forward mode to evaluate the gradient. The GRAD function can be called by C programs.

(27) The newest version (5.1) of **MAPLE** [Mona91a] can perform automatic differentiation. In Maple, functions are expressed as Maple procedures. The Maple D routine produces Maple procedures that evaluate single derivative values for functions described by other Maple procedures. The Maple *optimize* routine can then be used to optimize these procedures. Procedures for derivative values of arbitrary order and with respect to arbitrary independent variables can be generated. These procedures calculate derivatives in the forward mode. Maple procedures can be converted into FORTRAN or C subroutines.

(28) The **NR1** package [Rost91a] is a symbolic AD tool. NR1 is implemented in LISP within the Reduce Computer Algebra system. NR1 processes a Reduce procedure that satisfies certain restrictions and produces an array of Reduce expressions. These Reduce expressions calculate the gradient of the function in the reverse mode. This array of Reduce expressions can be converted to FORTRAN by the Reduce system. Currently the NR1 package is purely a research AD tool.

(29) The suite of FORTRAN programs **SVALAQ** (*Self-Validating Adaptive Quadrature*) [Corl87a] contain elements that allow it to be classified as a symbolic AD tool. This suite of programs produces guaranteed bounds on the values of certain definite integrals by using automatic differentiation. SVALAQ takes as input a single FORTRAN expression (no longer than 80 characters) for the integrand. The expression is parsed and a code list is generated. From the code list, quadrature rules for some special weight functions are activated. The code list is interpreted and interval-valued Taylor series are calculated in the forward mode. The calculated Taylor series are of arbitrary length.
SVALAQ is proprietary to IBM, and has not been released publicly.

5 Conclusion. In an effort to clarify the current state of AD technology, we presented a taxonomy of automatic differentiation tools in §3, and applied that taxonomy to a survey of twenty-nine AD tools in §4. We hope that this paper will be a valuable resource for those interested in applying automatic differentiation to their particular application.

Interested readers are directed to Appendix A for further information regarding the AD tools presented in §4. The appendix contains a list detailing the availability of numerous AD tools.

Acknowledgments. I thank Chris Bischof, George Corliss, Andreas Griewank, and James Lathrop for helpful suggestions in the preliminary stages of this paper. I also thank all of the many authors who provided information on their respective automatic differentiation tools.

Appendix A. A number of the automatic differentiation tools mentioned here are either commercially available, available as beta test versions to educational institutions, or available upon request. The following list details the availability of the various packages.

ADDS The ADDS package can be obtained by contacting Dr. Toshinobu Yoshida (yoshida @cs.gunma-u.ac.jp), Dept. of Computer Science, Gunma University, Kiryu, Gunma 376, Japan. This package should run on any system with Common LISP.

ADOL-C The source code and documentation for ADOL-C can be obtained via electronic mail by contacting Andreas Griewank (griewank@antares.mcs.anl.gov), c/o Argonne National Labs, Div. of Math & Computer Science, 9700 S. Cass Avenue, Argonne, IL 60439. (312) 972-6722.

AMPL Currently, most universities can obtain a beta-test version of the AMPL translator. Licensing for non-academic purposes is being planned. The AMPL translator should run on any system with a C compiler. For further details, contact David M. Gay (dmg@research.att.com), AT&T Bell Laboratories, Room 2C-463, 600 Mountain Avenue, Murray Hill, NJ 07974-2070, U.S.A., (908) 582-5623.

Package	Mode	Source	Avail.[1]	Deriv+[2]
Elemental				
WCOMP	F	FORTRAN	✓	✓
FEED	F	FORTRAN		✓
MJ1	F	PASCAL	✓	✓
Integral				
FOXY	F	FOXY	✓	✓
FM/FAD	R	DIFALG	✓	
AMPL	B	AMPL	✓	
ADDS	B	ADDS	✓	✓
Extensional				
ATOMFT	F	FORTRAN exp.	✓	✓
DAFOR	F	FORTRAN	✓	✓
FC	F	FORTRAN	✓	✓
GRAD	F	FORTRAN sub.	✓	✓
JAKEF	R	FORTRAN sub.	✓	
DAPRE	B	FORTRAN sub.	✓	✓
GRESS	B	FORTRAN	✓	
PADRE2	B	FORTRAN sub,	✓	✓
PCOMP	B	FORTRAN sub.	✓	✓
Operational				
GC1	F	ADA	✓	✓
GC2	F	PASCAL-SC	✓	✓
MJ2	F	C++	✓	✓
MXYZPTLK	F	C++	✓	✓
RL1	F	C++	✓	✓
RN1	F	PASCAL-SC		✓
BC1	R	ADA	✓	✓
ADOL-C	B	C++	✓	✓
Symbolic				
AD	F	Symbolic		✓
MATLAB	F	Symbolic	✓	✓
SVALAQ	F	FORTRAN exp.		✓
MAPLE	F	Maple procedure	✓	✓
NR1	R	Reduce procedure		✓

Table 1: AD tools divided by class.

[1] The details regarding the availability of the checked packages are in Appendix A.

[2] Check marks indicate that those packages can generate higher than first-order derivatives.

ATOMFT This can be obtained from George Corliss (georgec@boris.mscu.mu.edu) Dept. of Math., Marquette University, Milwaukee, WI 53233. Any system with a Fortran 77 compiler should be sufficient. The preferred form of distribution is e-mail or a 5 1/2" floppy, but other forms can be negotiated.

BC1 The BC1 package is available solely for educational and research purposes (no commercial or military use). The source for BC1 is written in ADA, and can be obtained by contacting Bruce Christianson (comqbc@hatfield.ac.uk), Numerical Optimisation Centre, Hatfield Polytechnic, Hatfield Hertfordshire AL10 9AB, England.

DAFOR The tool is available from Martin Berz for non-commercial scientific use. It runs on any standard FORTRAN 77 system and has been used on VAX, SUN, CRAY, CYBER, IBM PC, and MACINTOSH machines. Code can be obtained via BITNET or on tape or floppy by contacting BERZ@MSUNSCL.BITNET.

DAPRE Both Sun/Unix and VAX/VMS versions of DAPRE exist. For more information, contact NAG Ltd., Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, U.K. (telephone (+44)-865-511245 or fax (+44)-865-310139) or NAG Inc., 1400 Opus Place, Suite 200, Downers Grove, IL 60515-5702, USA (telephone (+1)-708-971-2337 or fax (+1)-708-971-2706).

FC FORTRAN CALCULUS is a commercially available product of the Digital Calculus Corporation. For more information, contact Davis Allen, VP Marketing, Digital Calculus Corp., 2615 Pacific Coast Hwy. Suite 215, Hermosa Beach, CA 90254, (213)-318-8822, (213)-318-2142 (FAX).

FM/FAD Those interested in obtaining the FM/FAD system should contact Dr. Vladimir P. Mazourik, Department of Application Software for Personal Computers, Computing Center, USSR Academy of Sciences, Vavilova str. 40, Moscow 117967, USSR, 135-6161.

FOXY The tool is available from Martin Berz for non-commercial scientific use after signing a short user registration including a non-proliferation agreement. It runs on any standard FORTRAN 77 system and has been used on VAX, SUN, CRAY, CYBER, IBM PC, and MACINTOSH machines. Code can be obtained via BITNET or on tape or floppy by contacting BERZ@MSUNSCL.BITNET.

GC1 The GC1 package of interval Taylor operators is available from George Corliss (georgec @boris.mscs.mu.edu), Dept. of Math., Marquette University, Milwaukee, WI, 53233. The source code can be obtained directly via e-mail. It is also available on 3 1/2" or 5 1/2" floppies, or other media by request.

GC2 The GC2 package of real and interval Taylor operators can be obtained by e-mail or on floppy disks by contacting George Corliss (georgec@boris.mscs.mu.edu), Dept. of Math., Marquette University, Milwaukee, WI, 53233, or Louis Rall (rall@ math .wisc.edu), Dept. of Math., University of Wisconsin - Madison, Madison, WI 53706.

GRAD The GRAD precompiler is free and can be obtained by contacting Oscar Garcia (garciao@mof.govt.nz or garciao%mof.govt.nz@uunet.uu.net), Forest Research Institute, Private Bag 3020, Rotorua, New Zealand. GRAD was written in APL. Two versions of GRAD are available, one for STSC APL*PLUS (which also runs with Pocket APL), and the other for the public domain I-APL interpreter .

GRESS GRESS can be obtained by contacting the Radiation Shielding and Information Center (RSIC), Oak Ridge National Laboratory, MS-6362, P.O. Box 2008, Oak Ridge TN 37831-6362, (615)-574-6176. RSIC charges a nominal recovery cost for GRESS. For more detailed information about GRESS, contact James Horwedel (jqh@ctd.ornl.gov), Computing and Telecommunications Division, Oak National Laboratory, Oak Ridge, TN 37831.

JAKEF This package currently resides on NETLIB.[3] Both the documentation and the sources for JAKEF can be obtained from NETLIB. JAKEF was written using FORTRAN-77 and has been ported to a variety of different machines.

MATLAB The source or executable code for the GRAD function in MATLAB is available from David R. Hill, Mathematics Department, Temple University, Philadelphia, Pa. 19122, or Lawrence C. Rich, The Mathematical Software Group, 54 Sequoia Drive, Newtown, Pa. 18940.

MJ1 & MJ2 The source code for both of these tools is available at no charge, and can be obtained by contacting Max E. Jerrell, College of Business Admin., Northern Arizona University, C.U. Box 15066, Flagstaff, AZ 86011. MJ1 was written using Turbo Pascal v5.5 and MJ2 was written using Turbo C++ v1.0.

MXYZPTLK The source code and documentation for MXYZPTLK can be obtained by contacting Dr. Leo Michelotti (michelotti@adcalc.fnal.gov), c/o Fermi Lab., P.O. Box 500, Mail Station 345, Batavia, IL 60510. (708)-840-4956.

RL1 The source code for RL1 is available, subject to the approval of management at the Aerospace Corporation. Those interested should contact either Dan Kalman (kalman@aerospace.aero.org) or Robert Lindell (lindell@aerospace.aero.org), M1/102, The Aerospace Corporation, P.O. Box 92957, Los Angeles, CA 90009-2957

PADRE2 The source code and documentation for PADRE2 can be obtained by contacting Dr. Koichi Kubota (kubota@ae.keio.ac.jp), Dept of Adm. Engineering, Faculty of Science and Technology, Keio University, 3-14-1, Hiyoshi Kohuku-ku, Yokohama 223, Japan. PADRE2 is available under UNIX as uuencoded compressed tar-file via electronic mail, or on a 720Kbyte Floppy Disk under DOS.

PCOMP PCOMP was implemented and tested on VAX/VMS, HP-UNIX and MS-DOS systems, in the latter case by using two different compilers. Thus PCOMP should run on the majority of existing computer systems. The source code is distributed by Prof. K. Schittkowski (Klaus.Schittkowski@uni-bayreuth.de), Mathematisches Institut, Universität Bayreuth, 8580 Bayreuth, Germany, and can be obtained on a MS-DOS diskette or a tape.

WCOMP The WCOMP and UCOMP packages can be used on any system with FORTRAN 77. They are available as part of the MATH77 library. The MATH77 library and its manual are available at a nominal price from COSMIC (Computer Software Management and Information Center), The University of Georgia, 382 East Broad Street, Athens, GA 30602. Persons with a research interest in the packages should contact Charles Lawson directly (Charles_LAWSON@jems.jpl.nasa.gov).

References

[Adam69a] D. S. Adamson and C. W. Winant, *A SLANG simulation of an initially strong shock wave downstream of an infinite area change*, in Proceedings of the Conference on Applications of Continuous-System Simulation Languages, 1969, pp. 231–240.

[Berz87a] M. Berz, *The differential algebra FORTRAN precompiler DAFOR*, Technical Report AT–3 TN–87–32, Los Alamos National Laboratory, Los Alamos, NM, 1987.

[Berz89a] ——, *Differential algebraic description of beam dynamics to very high order*, Particle Accelerators, 24 (1989), p. 109.

[Berz90c] ——, *Computational aspects of design and simulation: COSY INFINITY*, Nuclear Instruments and Methods, A298 (1990), pp. 473 +.

[Berz90d] ——, *COSY INFINITY, an arbitrary order general purpose optics code*, in Computer Codes and the Linear Accelerator Community, Los Alamos LA–11857–C, 1990,

[3] Netlib is a network service for the distribution of mathematical software via electronic mail. To receive more information, send the mail message "`send index`" to `netlib@research.att.com`.

pp. 137 +.

[Berz90b] ——, *COSY INFINITY version 3 reference manual*, Technical Report MSUCL – 751, National Superconducting Cyclotron Lab., Michigan State University, East Lansing, Mich. 48824, 1990.

[Berz90a] ——, *Differential algebra precompiler version 3 — Reference manual*, Technical Report MSUCL – 755, National Superconducting Cyclotron Laboratory, Michigan State University, East Lansing, Mich. 48824, 1990.

[Chan82a] Y. F. Chang and G. F. Corliss, *Solving ordinary differential equations using Taylor series*, ACM Trans. Math. Software, 8 (1982), pp. 114–144.

[Chri90a] B. D. Christianson, *Automatic Hessians by reverse accumulation*, Technical Report NOC TR228, The Numerical Optimisation Center, Hatfield Polytechnic, Hatfield, U.K., April 1990.

[PROS77a] Control Data Corporation, *PROSE – A General Purpose Higher Level Language*, 1977. Procedure Manual (Pub. No. 84003000 Rev. B), Calculus Operations Manual (Pub. No. 84003200 Rev. A), Calculus Applications Guide (Pub. No. 84000170 Rev A).

[Corl91a] G. F. Corliss, *Overloading point and interval Taylor operators*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. F. Corliss, eds., SIAM, Philadelphia, Penn., 1991, pp. 139–146.

[Corl84a] G. F. Corliss and L. B. Rall, *Automatic generation of Taylor series in Pascal-SC: Basic operations and applications to differential equations*, in Trans. of the First Army Conference on Applied Mathematics and Computing (Washington, D.C., 1983), ARO Rep. 84-1, U. S. Army Res. Office, Research Triangle Park, N.C., 1984, pp. 177–209.

[Corl87a] ——, *Adaptive, self-validating quadrature*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. 831–847.

[Corl91b] ——, *Computing the range of derivatives*, in IMACS Annals on Computing and Applied Mathematics, E. Kaucher, S. M. Markov, and G. Mayer, eds., vol. 12 of IMACS Annals on Computing and Applied Mathematics, J. C. Baltzer AG, Basel, 1991, pp. 195–212.

[Flan91b] H. Flanders, *Response to electronic mail survey.* Personal communication, March 1991.

[Four90a] R. Fourer, D. M. Gay, and B. W. Kernighan, *A modeling language for mathematical programming*, Management Science, 36 (1990), pp. 519–554.

[Garc91a] O. García, *A system for the differentiation of Fortran code and an application to parameter estimation in forest growth models*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. F. Corliss, eds., SIAM, Philadelphia, Penn., 1991, pp. 273–286.

[Gayd91b] D. M. Gay, *Response to electronic mail survey.* Personal communication, March 1991.

[Grie89a] A. Griewank, *On automatic differentiation*, in Mathematical Programming: Recent Developments and Applications, M. Iri and K. Tanabe, eds., Kluwer Academic Publishers, 1989, pp. 83–108. Also appeared as Preprint MCS–P10–1088, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., October 1988.

[Grie92a] A. Griewank, D. Juedes, J. Srinivasan, and C. Tyner, *ADOL-C, a package for the automatic differentiation of algorithms written in C/C++*, ACM Trans. Math. Software, (to appear). Also appeared as Preprint MCS–P180–1190, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1990.

[Hill91a] D. R. Hill and L. C. Rich, *Automatic differentiation in MATLAB*, Applied

Numerical Mathematics, (to appear).

[Hill85a] K. E. Hillstrom, *Users guide for JAKEF*, Technical Memorandum ANL/MCS–TM–16, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1985.

[Horw90a] J. E. Horwedel, R. J. Raridon, and R. Q. Wright, *Sensitivity analysis of AIRDOS–EPA using ADGEN with matrix reduction algorithms*, Technical Memorandum ORNL/TM 11373, Martin Marietta Energy Systems, Inc., Oak Ridge National Laboratory, Oak Ridge, Tenn. 37830, 1989.

[Horw88a] J. E. Horwedel, B. A. Worley, E. M. Oblow, and F. G. Pin, *GRESS version 1.0 users manual*, Technical Memorandum ORNL/TM 10835, Martin Marietta Energy Systems, Inc., Oak Ridge National Laboratory, Oak Ridge, Tenn. 37830, 1988.

[Horw89a] J. E. Horwedel, R. Q. Wright, and R. E. Maerker, *Sensitivity analysis of EQ3*, Technical Memorandum ORNL/TM 11407, Oak Ridge National Laboratory, Oak Ridge, Tenn. 37830, 1990.

[Irim87a] M. Iri and K. Kubota, *Methods of fast automatic differentiation and applications*, Research Memorandum RMI 87 – 02, Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo, 1987.

[Iwat84a] N. Iwata, *Automatization of the computation of partial derivatives*, master's thesis, Graduate School, University of Tokyo, 1984. (In Japanese).

[Jerr89a] M. Jerrell, *Automatic differentiation and function minimization in C++*, in Proceedings of OOP–SLA, ACM Press, 1989, pp. 18–24.

[Jerr89b] ——, *Automatic differentiation using almost any language*, ACM SIGNUM Newsletter, (1989), pp. 2–9.

[Jerr90a] ——, *Automatic differentiation using C++*, Journal of Object Oriented Programming, (1990), pp. 17–24.

[Jued90a] D. Juedes and A. Griewank, *Implementing automatic differentiation efficiently*, Technical Memorandum ANL/MCS–TM–140, Mathematics and Computer Sciences Division, Argonne National Laboratory, Argonne, Ill., 1990.

[Krin84a] B. Krinsky and J. Thames, *The structure of synthetic calculus*, in Proceedings of the International Workshop on High-Level Computer Architecture, University of Maryland, 1984.

[Kubo90a] K. Kubota and M. Iri, *PADRE2, version 1 — User's manual*, Research Memorandum RMI 90–01, Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo, 1990.

[Laws71a] C. L. Lawson, *Computing derivatives using W-arithmetic and U-arithmetic*, Internal Computing Memorandum CM–286, Jet Propulsion Laboratory, Pasadena, CA 91105, September 1971.

[Liep90a] M. Liepel and K. Schittkowski, *PCOMP: A FORTRAN code for automatic differentiation*, Report No. 254, DFG Schwerpunktprogramm Anwendungsbezogene Optimierung und Optimale Steuerung, Mathematisches Institut, Universität Bayreuth, D-8580 Bayreuth, Germany, 1990.

[Lind91a] R. Lindell, *Response to electronic mail survey.* Personal communication, March 1991.

[Math89a] MATH, *MATH77, Release 3.0, A library of mathematical subprograms for FORTRAN 77*, Internal Document D–134, Rev. B, Jet Propulsion Laboratory, Pasadena, Calif. 91105, May 1989. Also available as Program No. NPO–18120 from COSMIC (Computer Software Management and Information Center), The University of Georgia, Athens, GA.

[Mazo91a] V. Mazourik, *Integration of automatic differentiation into a numerical library for PC's*, in Automatic Differentiation of Algorithms: Theory, Implementation, and

Application, A. Griewank and G. F. Corliss, eds., SIAM, Philadelphia, Penn., 1991, pp. 286–293.

[McCu69a] J. D. McCully, *The Q approach to problem solving*, in Proceedings of FJCC 69, AFIPS, 1969, pp. 691–699.

[Mich90a] L. Michelotti, *MXYZPTLK: A practical, user-friendly C++ implementation of differential algebra: User's guide*, Technical Memorandum FN–535, Fermi National Accelerator Laboratory, Batavia, Ill. 60510, January 1990.

[Mona91a] M. Monagan, *Response to electronic mail survey.* Personal communication, March 1991.

[Neid91a] R. D. Neidinger, *An efficient method for the numerical evaluation of partial derivatives of arbitrary order*, ACM Trans. Math. Software, (to appear).

[Pfei87a] F. W. Pfeiffer, *Automatic differentiation in PROSE*, ACM SIGNUM Newsletter, 22 (1987), pp. 1–8.

[Rall81a] L. B. Rall, *Automatic Differentiation: Techniques and Applications*, vol. 120 of Lecture Notes in Computer Science, Springer Verlag, Berlin, 1981.

[Rost91a] N. Rostaing and M. Gaetano, *Response to electronic mail survey.* Personal communication, March 1991.

[Spee80a] B. Speelpenning, *Compiling Fast Partial Derivatives of Functions Given by Algorithms*, PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana-Champaign, Ill. 61801, January 1980.

[Step91a] B. R. Stephens and J. D. Pryce, *DAPRE: A differentiation arithmetic system for FORTRAN*, Technical Report ACM–91–3, Royal Military College of Science, Shrivenham, U.K., 1991.

[Tesf91a] L. Tesfatsion, *Automatic evaluation of higher-order partial derivatives for nonlocal sensitivity analysis*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. F. Corliss, eds., SIAM, Philadelphia, Penn., 1991, pp. 157–165.

[Tham69a] J. M. Thames, *SLANG, a problem-solving language for continuous-model simulation and optimization*, in Proceedings of the ACM 24th National Conf., ACM, 1969.

[Tham75a] ——, *Computing in calculus*, Research/Development, (1975), pp. 24–30.

[Tham82a] ——, *The evolution of synthetic calculus: A mathematical technology for advanced architecture*, in Proceedings of the International Workshop on High-Level Language Computer Architecture, University of Maryland, 1982.

[Tham89a] ——, *FORTRAN CALCULUS: A New Implementation of Synthetic Calculus*, Digital Calculus Corp., Torrance, Calif. 90505, 1989.

[Tham91a] ——, *Synthetic calculus: A paradigm for mathematical program synthesis*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. F. Corliss, eds., SIAM, Philadelphia, Penn., 1991, pp. 263–272.

[Yosh89a] T. Yoshida, *Automatic derivative derivation system*, Transactions of Information Processing Society of Japan, 30 (1989), pp. 799–806. (In Japanese).